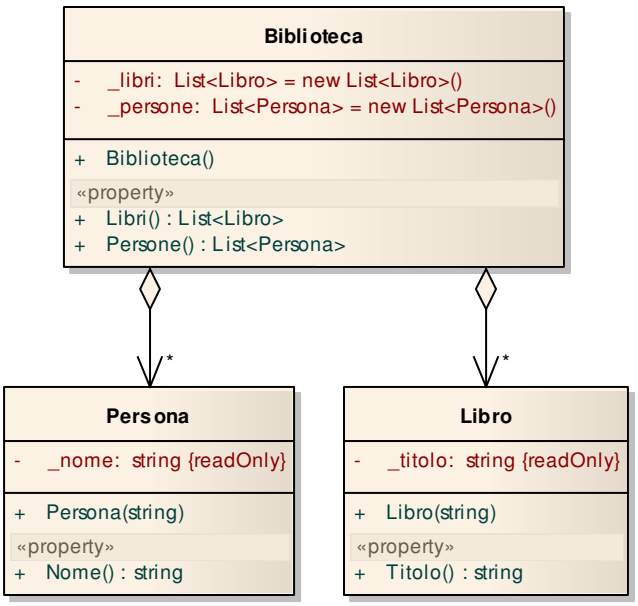
Laboratorio 1

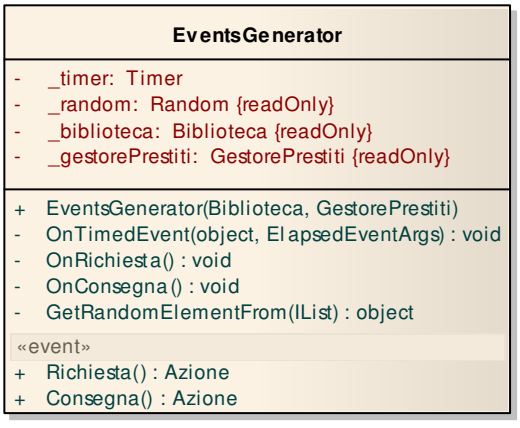
Passo 1 – Creare un progetto di tipo **`ConsoleApplication`** e di nome **`Lab1`**. Definire le classi **`Libro`**, **`Persona`** e **`Biblioteca`**, come descritte dal diagramma UML di figura. Nel costruttore della classe **`Biblioteca`** popolare la biblioteca con 5 libri e 5 persone. Nei costruttori delle classi **`Libro`** e **`Persona`** non si deve accettare come argomento il valore **`null`** o una stringa vuota.



Passo 2 – Definire una nuova classe **`GestorePrestiti`** (ved. il corrispondente diagramma UML), che ha la responsabilità di gestire il prestito dei libri. La classe contiene un dizionario in cui ogni chiave è un libro correntemente prestato e il corrispondente valore è la persona cui tale libro è stato prestato.
`LibriInPrestito` restituisce l'elenco di tutti i libri che risultano prestati al momento della chiamata.

`IsLibroDisponibile` restituisce **`true`** se il libro passato come argomento non risulta in prestito.
`PossessoreLibro` restituisce la persona che ha in prestito il libro passato come argomento, oppure **`null`** se il libro non risulta in prestito.
`GestisciRichiesta` verifica se la richiesta di prestito di un libro da parte di una persona è effettuabile e, in caso affermativo, effettua l'operazione – in tutti i casi esegue una stampa a *console*. **`GestisciConsegna`**, dopo aver verificato la correttezza degli argomenti passati, effettua l'operazione di consegna di un libro.
Si noti che il **`GestorePrestiti`** è del tutto disaccoppiato dalla biblioteca di cui gestisce i prestiti. Pertanto, per verificarne il corretto funzionamento, nel **`Main`** è sufficiente creare un **`GestorePrestiti`** più alcuni libri e alcune persone e invocare in modo appropriato i vari metodi.

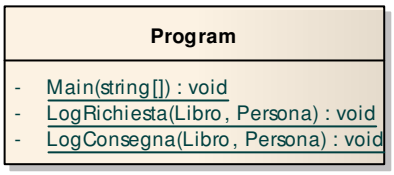
Passo 3 – Per simulare il funzionamento di una biblioteca, definire la classe **`EventsGenerator`** (ved. il corrispondente diagramma UML), che ha la responsabilità di generare in modo casuale richieste e consegne di libri. In particolare, la classe è in grado di generare gli eventi **`Richiesta`** e **`Consegna`** di tipo **`Azione`**. Il **`delegate Azione`** (da definire) accetta come argomenti un **`Libro`** e una **`Persona`** e non restituisce nulla.



Laboratorio 1

La classe **`EventsGenerator`**, oltre a contenere i riferimenti a una biblioteca, a un gestore prestiti e a un generatore di numeri casuali (**`_random`**), contiene un **`Timer`** (più precisamente un **`System.Timers.Timer`**) che permette di eseguire del codice a intervalli regolari. Nel costruttore di **`EventsGenerator`**, creare un'istanza di **`Timer`** e fare in modo che generi un evento ogni secondo e che tale evento sia collegato all'*handler* **`OnTimedEvent`** (vedere l'esempio nella documentazione della classe **`Timer`**). **Attenzione**: perché l'applicazione funzioni correttamente, è necessario abilitare il timer (proprietà **`Enabled`**) e inserire in fondo al **`Main`** le istruzioni di scrittura e di lettura a *console* riportate nell'esempio sopra citato. Verificare il funzionamento inserendo una semplice stampa a console nell'*handler*.

Il metodo **`OnTimedEvent`** deve invocare casualmente i metodi **`OnRichiesta`** e **`OnConsegna`** che a loro volta faranno scattare in modo opportuno i corrispondenti eventi. A tal fine, definire l'enumerativo **`TipoEvento`** annidato nella classe **`EventsGenerator`** (ved. il corrispondente diagramma UML) che elenca i tre possibili eventi che possono verificarsi ogni volta che scatta il *timer*. In **`OnTimedEvent`**, generare a caso uno dei valori dell'enumerativo (mediante il servizio **`GetRandomElementFrom`**) e invocare il metodo corrispondente (se il valore è **`Nop`**, non eseguire nulla).
Il metodo **`GetRandomElementFrom`** deve accettare come argomento una collezione che implementa **`IList`** e deve restituire uno degli elementi contenuti nella collezione, scelto a caso utilizzando in modo opportuno **`_random`**, oppure **`null`**, se la collezione è vuota. Verificare il corretto funzionamento del metodo, utilizzando un *array* di interi, una lista di interi e la collezione dei valori interi associati a un enumerativo (si sfrutti il metodo **`Enum.GetValues`**).
Infine, realizzare i metodi **`OnRichiesta`** e **`OnConsegna`** in modo che facciano scattare gli eventi corrispondenti, dopo aver generato con il metodo **`GetRandomElementFrom`** libri e persone coinvolte.
Nel **`Main`**, collegare gli eventi sia al gestore dei prestiti, sia ai metodi **`LogRichiesta`** e **`LogConsegna`** (ved. il diagramma UML) che devono contenere semplici stampe a *console*.



Passo 4 (opzionale) – Modificare il gestore dei prestiti in modo che accetti e soddisfi le prenotazioni di libri non disponibili.
Ogni persona interessata a un libro non disponibile deve essere inserita in una lista di attesa. Quando un libro viene riconsegnato e quindi diventa disponibile, il libro deve essere assegnato alla prima persona nella sua lista di attesa.
Attenzione: la stessa persona non deve comparire più di una volta nella lista di attesa dello stesso libro; inoltre, la persona che consegna un libro non deve riavere immediatamente lo stesso libro se ci sono altre persone in lista – cioè può essere nella lista, ma non in prima posizione.